COBB *Tips & techniques for MS-DOS and PC-DOS*

July 1990
Vol. 1 No. 3

Inside DOS™

An introduction to Edlin—DOS' text editor

In the May issue of *Inside DOS*, we showed you how to use the COPY CON command to create a batch file ("An Introduction to Batch Files"). Although COPY CON is fine for creating short, simple batch files, you will want to use a text editor to create and edit longer, more complex batch files.

Fortunately, DOS provides a simple text editor called *Edlin*. Although Edlin doesn't possess many of the advanced features offered by commercial word processors, Edlin is effective for creating and editing your batch files. If you want, you can even use Edlin to write simple notes, letters, and memos.

In order to use Edlin, DOS must be able to find the file EDLIN.EXE. If your computer is equipped with a hard disk, make sure your computer's path includes the directory containing your DOS files, as we explained in the article "Using the PATH Command to Access External Commands" in the May issue. If your computer is not equipped with a hard disk, place your DOS diskette in drive A and start Edlin from the A> prompt.

Creating a new text file

To create a new text file with Edlin, simply issue the command

```
edlin filename
```

where *filename* is the name you assign to the file you are creating. For example, if you want to create a batch file named LETTER.TXT, simply issue the command

```
C>edlin letter.txt
```

Immediately, DOS will respond

```
New file
*_
```

The message *New file* tells you that DOS is creating a new file. The asterisk (*) is the Edlin prompt, which tells you that Edlin is ready to accept an Edlin command, just as the C> or A> prompt tells you DOS is ready to accept a DOS command.

Entering lines

Of course, once you've opened a new file, all you can do is insert new lines. To do this, type the letter *i* (for *Insert*) and press ↵, like this:

```
*i
   1:*_
```

As you can see, Edlin responds by displaying the number *1*, followed by a colon, an asterisk, and the cursor. The number 1 tells you that you are working on the first line in the file. At this point, just start typing the text you want to appear on line 1, like this:

```
   1:*Dear Homer,_
```

If you make a mistake, you can press the [Backspace] key to back up and correct your error. When you're finished working on line 1, press ↵ to bring up the prompt for line 2:

```
   2:*_
```

Notice that this prompt contains the new line number, an asterisk, and the cursor. At this point, type the text you want to appear in line 2. Continue entering text line by line until you're finished, like this:

```
   2:*I know this comes as a surprise,
   3:*but I've left you for another man.
   4:*He's not as smart as you, but he's
   5:*very good looking and rich.
   6:*_
```

Continued on page 9

IN THIS ISSUE

COBB

Inside DOS™

Inside DOS (ISSN 1049-5320) is published monthly by The Cobb Group, Inc., 9420 Bunsen Parkway, Suite 300, Louisville, KY 40220. Domestic subscriptions: 12 issues, $39. Foreign: 12 issues, $59. Single issues: $4 domestic, $6.50 foreign. Send subscriptions, fulfillment questions, and requests for bulk orders to Customer Relations, 9420 Bunsen Parkway, Suite 300, Louisville, KY 40220, or call toll free 1-800-223-8720. Address correspondence and special requests to The Editor, *Inside DOS*, at the address above.

POSTMASTER: Send address changes to The Cobb Group, Inc., P.O. Box 35160, Louisville, KY 40232. Second class postage is pending in Louisville, KY.

Editor-in-Chief:	Mark W. Crane
Contributing Editors:	Van Wolverton
	Chris DeVoney
	Tim Landgrave
Editing:	Jim Welp
	Martha A. Clayton
Production:	Elayne Noltemeyer
Design:	Karl Feige
Publisher:	Douglas Cobb

Conventions

To avoid confusion, we would like to explain a few of the conventions used in *Inside DOS*.

As you probably know, Microsoft has released several versions of DOS. Although, most of the articles in this journal apply to all versions of DOS since 2.00, we'll sometimes point out a particular command or technique that requires a particular version. For simplicity, we'll refer to all 4.xx versions as Version 4.

Typically, we'll use DOS's default prompt, C>, to represent the DOS prompt in our examples. If you have defined a custom DOS prompt, just assume that the C> represents your prompt.

When we instruct you to type something, those characters usually appear on a separate line along with the DOS prompt. The characters you type will appear in color, while the characters DOS displays will appear in black.

Occasionally, we won't display the command you type on a separate line. In these cases, we'll display the characters you type in italics. For example, we might say, "issue the command *dir *.txt* at the DOS prompt." Although DOS is not sensitive to capitalization, we'll always display the characters you type in lowercase.

When we refer to a general DOS command (not the command you actually type at the DOS prompt), we'll display that command name in capital letters. For example, we might say, "You can use either the COPY or XCOPY command to transfer files from one disk to another."

Many commands accept parameters that specify a particular file, disk drive, or other option. When we show you the form of such a command, its parameters will appear in Italics. For example, the form of the COPY command is:

```
copy  file1  file2
```

where *file1* and *file2* represent the names of the source file and the target file, respectively.

The names of keys, such as [Shift], [Ctrl], and [F1], appear in brackets. The [Enter] key is represented by the symbol ↵. When two keys must be pressed simultaneously, those key names appear side by side, as in [Ctrl][Break] or [Ctrl]z.

LETTERS

Using MORE with a write-protected diskette

Q In the May issue of *Inside DOS*, you explained how to use the MORE command (see "Displaying the Contents of a Text File") to view the contents of a file one screen at a time. Unfortunately, the MORE command doesn't work when the file you want to view is stored on a write-protected diskette.

For instance, suppose I want to view the contents of the file README.TXT, which is stored on the write-protected diskette in drive A. To do this, I issue the command

```
A>type  readme.txt  |  more
```

However, instead of displaying the contents of the file, DOS responds

```
Write protect error writing drive A
Abort, Retry, Fail?
```

What's the problem?

Harley E. Beard
Everett, Washington

A Good question, Harley. Here's what's going on: Whenever you use the | character to pipe a command to another command, DOS creates a temporary file on the default drive to store the first command's output. Then, DOS "feeds" that temporary file to the second command and erases the temporary file before returning to the DOS prompt.

In the example you presented above, you issued the command

```
A>type  readme.txt  |  more
```

which pipes the command *type readme.txt* to the command *more*. As soon as you issued this command, however, DOS tried to create a temporary file to store the output of the command *type readme.txt*. Since the default drive is A, and drive A contains a write-protected diskette, DOS displayed an error message.

Fortunately, there are two ways to solve this problem and view the contents of a file on a write-protected diskette. First, you can change the default drive to a drive that contains an unprotected diskette, then issue the command from there. For instance, if your computer is equipped with a drive C, you could issue the commands

```
A>c:
C>type  a:readme.txt  |  more
```

Continued on back page

Sending a form feed to your printer

The idea for this article was submitted by Jim Bauman. Jim is the owner of Cogent Data Systems, Inc., in Ridgewood, New Jersey.

If you are like most people who work with a printer, you often need to send a form feed to your printer. This involves pressing the printer's On Line button to take the printer off line, then pressing the Form Feed button, and finally pressing the On Line button again to put the printer back on line. Executing this series of steps can sometimes be a real pain—especially if your computer isn't located near the printer.

Fortunately, you can use a simple technique to send a form feed to your printer from DOS. In this article, we'll explain the technique, and show you how to create a batch file that sends a form feed to the printer.

The ECHO command

The ECHO command is the tool you'll use to send a form feed command to the printer. As you may know, the ECHO command normally sends messages to your screen. For example, if you issue the command

```
C>echo Hello
```

DOS will display the message *Hello* on your screen. However, if you want to send the message *Hello* to your printer instead of the screen, you can redirect the command's output to the printer by appending > *prn* to the command, like this:

```
C>echo Hello > prn
```

By the way, the device name PRN typically represents DOS's default parallel port—LPT1. If your printer is connected to port LPT1, the command *echo Hello > prn* will have the same result as the command *echo Hello > lpt1*. If your printer is connected to port LPT2 or LPT3, you'll want to use the device names LPT2 or LPT3 instead of PRN. For example, to send the message *Hello* to the printer connected to port LPT3, issue the command

```
C>echo Hello > lpt3
```

Echoing a form feed

Now that you know how to use the ECHO command to send messages to your printer, let's learn how to send a form feed command. The universal code that tells the printer to execute a form feed is [Ctrl]L. This code should work with *any* printer, including an ancient dot-matrix printer and a sophisticated Hewlett-Packard LaserJet.

To send a form feed command to your printer, type *echo* followed by a space, press [Ctrl]L, type a space and the characters > *prn*, and press ↵ like this:

```
C>echo ^L > prn
```

Notice that DOS represents the [Ctrl]L with the characters ^*L* on the command line. As long as your printer is on line when you issue this command, it will immediately execute a form feed and advance to the next page.

Creating an FF.BAT file

Although you can issue the command

```
C>echo ^L > prn
```

each time you need to send a form feed to your printer, it's much easier to create a file named FF.BAT that stores this ECHO command. That way, whenever you want to send a form feed, you can simply type *ff* and press ↵.

To create the file FF.BAT, just type

```
C>copy con ff.bat
echo ^L > prn
^Z
        1 File(s) copied

C>_
```

Now, if you place the file FF.BAT into a directory on the system path, you can send a form feed to your printer whenever you want by issuing the command

```
C>ff
```

Sending a form feed from within a batch file

If you write batch files that send data to your printer, you might want to include a form feed command at the end of your last print command to advance the printer to the next page. For example, suppose you've created a batch file that contains the following three commands:

```
type report1.txt > prn
type report2.txt > prn
type report3.txt > prn
```

Now, to finish the job right, you'll want to add a fourth command to your batch file that issues a form feed to the printer:

```
echo ^L > prn
```

That way, you can retrieve your report from the printer without having to use the printer's On Line and Form Feed buttons.

What are the . and .. entries in a subdirectory?

If you're like most computer users, you've occasionally wondered why DOS automatically places the entries . (single dot) and .. (double dot) in a subdirectory. For instance, suppose you create a new subdirectory called TEST by issuing the command

```
C:\>md test
```

Then, you move into that directory by typing

```
C:\>cd test
```

Finally, you list the contents of that directory by typing

```
C:\TEST>dir
```

Instead of finding the directory empty, DOS will display the following:

```
 Volume in drive C has no label
 Directory of   C:\TEST

.            <DIR>          7-15-90   11:28a
..           <DIR>          7-15-90   11:28a
          2 File(s)    11315200 bytes free

C:\TEST>_
```

What are the two entries . and ..? Why do they always appear in a subdirectory? In this article, we'll answer these questions, and show you how you can use these entries to your advantage.

By the way, all the DOS prompts you'll see in this article include both the current drive and directory. To create this type of prompt on your system, issue the command *prompt pg*.

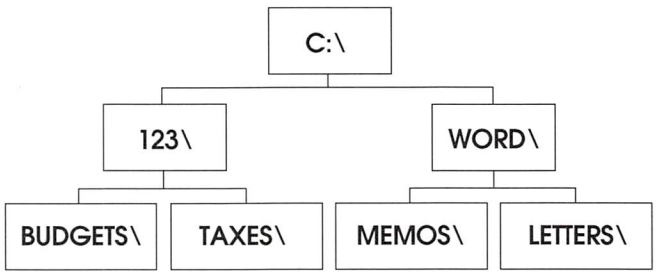
Parent and child directories

Understanding the concept of parent and child directories is essential to understanding the . and .. entries. To illustrate this concept, consider the directory structure shown in Figure A.

As you can see, there are two directories that branch off the C:\ root directory—123\ and WORD\. These directories are called the root's *child directories*, since they branch off the root directory. Additionally, the root directory is called the *parent directory* of the 123\ and WORD\ directories.

Just as 123\ and WORD\ are child directories of the root directory, MEMOS\ and LETTERS\ are child directories of the WORD\ directory. Conversely, the WORD\ directory is the parent of the MEMOS\ and LETTERS\ directories.

Figure A



We'll use this directory structure to illustrate a few basic directory concepts.

How DOS creates a directory

As soon as you issue the MD or MKDIR command, DOS sets aside disk space to hold the entries for the directory you've created. Additionally, DOS establishes a link into and out of this directory. In other words, DOS records the address of both the new subdirectory and its parent directory, so that it can enter and leave the new directory later on. Here's where the . and .. entries come into play.

The . entry

DOS uses the . (single dot) entry to store the new directory's address. Later, if you use . on the command line, DOS will refer to the address stored in the . entry to find where that directory exists on the hard disk. Unfortunately, you can't do much with the . entry on the command line. However, you can use the . entry along with the DEL or ERASE command to erase every entry in the current subdirectory. In other words, to delete every file in the current directory, you can issue the command

```
C:\TEST>del  .
```

instead of the command

```
C>\TEST>del  *.*
```

The .. entry

Just as DOS needs a . entry to record the address of each directory, it needs a .. (double dot) entry to record the address of the directory's parent. In other words, just as the . entry points to the new directory, the .. entry points one level up to the parent directory.

For instance, suppose you've set up the directory structure shown in Figure A, and you're currently working in the directory C:\123\TAXES. The .. entry in this directory points to its parent directory—C:\123. Similarly, the .. entry in the directory C:\123 points to its parent—C:\.

Stepping through directories

You'll find that the .. entry comes in handy in a variety of situations. For instance, the .. entry makes it easier to step through the subdirectories on your hard disk. If you are in the directory C:\123\TAXES, and you want to move up one level to the directory's parent (C:\123), you can use the command

```
C:\123\TAXES>cd  ..
C:\123>_
```

If you want, you can chain together the .. entry and a directory path to move up then down through the directory structure. For instance, to move from C:\123TAXES to C:\123BUDGETS, use the command

```
C:\123\TAXES>cd  ..\budgets
C:\123:BUDGETS>_
```

As you might guess, you can use the .. entry in other commands, too. For example, suppose you're in the directory C:\123\TAXES, and you want to copy the file INCOME.WK1 up one level to the parent directory (C:\123). To do this, you can use the command

```
C:\123\TAXES>copy  memo.wk1  ..
```

Shortening your path

Another situation in which the .. entry can come in handy is in your system path. Unfortunately, your system path can be no longer than 123 characters. If you've included so many directories in your path that you're bumping up against this limitation, you might want to use the technique we'll describe to reduce the number of characters in your path.

In order to take advantage of the following technique, you need to model your disk's directory structure after the one shown in Figure A. That is, you need to store the data for each of your applications in child directories off of the applications' directories. For instance, if you've installed Lotus 1-2-3 in the directory C:\123, you need to store all of your 1-2-3 worksheets in the data directories that branch directly off of the directory C:\123.

Once you have set up this type of directory structure, you don't need to list the full path name of each application directory on your hard disk. Instead, you can simply put a double dot (..) on the path to allow your application to find all of its program files. Here's why: As you use an application to open and save data files, the current directory will always be a child of the application directory. Consequently, when the application needs to access a program file in its directory, the path entry .. will tell the application to look in the current directory's parent directory, which will contain the application's program files.

For our sample directory structure in Figure A, then, we can use the command

```
C:\>path=c:\dos;..
```

to define our path instead of the command

```
C:\>path=c:\dos;c:\123;c:\word
```

Although the shorter PATH command only saves a few characters in this simple example, the savings will increase significantly if you've installed several applications on your hard disk.

Conclusion

DOS automatically includes two entries (. and ..) in every subdirectory you create. In this article, we have explained the purpose of these entries, and we have shown you a couple of tips so you can take advantage of them.

Contributing Editor Van Wolverton is author of the bestselling books Running MS-DOS *and* Supercharging MS-DOS.

Getting to know a very important file: AUTOEXEC.BAT

Have you ever installed a new program, then discovered when you restarted your system that it didn't work properly? Maybe a menu program didn't run, or you couldn't use your DOS commands from any directory. Do you have to run a special program in order to use a device attached to your system? Do you ever get tired of typing the same commands every time you start your system? If so, you need to look at (or create) a file named AUTOEXEC.BAT.

What is AUTOEXEC.BAT?

AUTOEXEC.BAT is a special batch file. As you've learned from other articles in *Inside DOS*, a batch file is simply a file that contains DOS commands and whose name ends with the extension BAT.

Each time you turn on your machine or restart DOS by pressing [Ctrl][Alt][Delete], DOS checks the root directory of the system disk to see if there is a file named AUTOEXEC.BAT; if it finds one, it carries out the commands in the file before turning the system over to you. (As you've probably figured out, AUTOEXEC is short for AUTOmatically EXECute.)

If there is no AUTOEXEC.BAT file in the root directory of the system disk, DOS prompts you for the date and time, even though it almost always knows the correct date and time by reading the system clock. If it finds AUTOEXEC.BAT, however, DOS doesn't prompt you for the date and time unless the file contains a DATE and TIME command. AUTOEXEC.BAT is one of only three files that DOS requires in the root directory of your system disk (the other two are CONFIG.SYS and COMMAND.COM; we'll cover these in a future issue of *Inside DOS*).

Starting with Version 4, the DOS system disk includes an AUTOEXEC.BAT file that starts the installation process that creates the startup and operating diskettes or copies the DOS files to your hard disk. Before Version 4, however, there was no AUTOEXEC.BAT file unless you created one yourself.

To create an AUTOEXEC.BAT file, you can use DOS' built in text editor—Edlin (see this month's lead article, "An Introduction to Edlin—DOS' text editor). If you want, however, you can create an AUTOEXEC.BAT file using your favorite word processor or text editor (make sure, though, that the word processor or text editor does not put any special formatting characters in the file). Finally, you can use the COPY CON command (which we have used on page 8) to create a simple AUTOEXEC.BAT file.

What should be in AUTOEXEC.BAT?

You can include any DOS commands in AUTOEXEC.BAT. As articles in earlier issues of *Inside DOS* have pointed out, any command or series of commands that you use frequently are candidates for a batch file. At a minimum, you should include PROMPT and PATH commands to make your system easier to use and tailor it to your needs.

The PROMPT Command

The PROMPT command tells DOS what to display when it returns to the command level and waits for you to type a command. If you don't specify anything else, DOS simply displays the current drive letter followed by a greater-than symbol; if you have a hard disk, that is C>.

You can include several items of information, as well as any characters you like, in the system prompt. Among these are the time, date, current directory, a backspace, and a new line, as shown in Table A. You specify the information or character you want by including a $ followed by a letter as a parameter for the PROMPT command. (For a complete list of the information you can include in your prompt, see the article "Creating a Custom Prompt with the PROMPT Command" in the May issue.

Table A

Information or Character	$+letter
Time	$t
Date	$d
Current directory	$p
Backspace	$h
New line	$_

Let's briefly look at a couple of useful forms of the PROMPT command. The following examples assume that the current drive is C, and the current directory is

\123\BUDGET. It's always useful to know where you are, so the following PROMPT command gives the most information for the least effort:

```
C>prompt $p$g
C:\123\BUDGET>_
```

If you want to include the date and time as well, try this:

```
C>prompt $d   $t$_$p$g
Tue   7-17-90     14:40:23.15
C:\123\BUDGET>_
```

If the tenths and hundredths of a second are too distracting, you can get rid of them with six backspaces (*$h*) following the *$t*:

```
C>prompt $d   $t$h$h$h$h$h$h$_$p$g
Tue   7-17-90     14:40
C:\123\BUDGET>_
```

Spend a few minutes designing a system prompt and put the PROMPT command in AUTOEXEC.BAT; you'll benefit each time you use DOS.

You can include any DOS commands in AUTOEXEC.BAT... At a minimum, you should include a PROMPT and PATH command.

The PATH Command

The PATH command tells DOS where to find a command file if it isn't located in the current directory. If you put the path name of the directories that contain your DOS files and your application programs in the command path, you can use any DOS command and most application programs from any directory. Otherwise, you must change to the directory that contains the DOS command files or the application program each time you use one of them.

Assume that you use Lotus 1-2-3, Microsoft Word, and Generic CAD in directories named 123, WORD, and CADD, and that the DOS files are in C:\DOS. This PATH command tells DOS where to find your program files:

```
path=c:\dos;c:\123;c:\word;c:\cadd
```

Again, do this just once by putting the PATH command in AUTOEXEC.BAT. When you add another application program to your system, simply add the new directory to the PATH command.

Other candidates for AUTOEXEC.BAT

In addition to putting PROMPT and PATH commands in AUTOEXEC.BAT, you can streamline your system by including commands that

- start any special programs (usually called *drivers*) for devices such as a mouse, external disk drive, or high-resolution display
- start any terminate-and-stay-resident programs (TSRs) such as Sidekick or ProKey
- send configuration commands to your printer
- copy files to a RAM disk
- change the directory to the one in which you usually start
- start a program you always use first, such as a menu system or an appointment calendar

Remember, these are just some of the possibilities; you'll want to include any command or series of commands you routinely type when you start your system.

Beware of helpful programs

When you install an application program, you are usually instructed to include in the command path the directory where the program is stored; many programs offer to do it for you by modifying AUTOEXEC.BAT.

It's usually best to make this change yourself, if you have the choice, because many applications simply add a PATH command that points to their directory to the end of AUTOEXEC.BAT. Because DOS replaces the path each time it finds a new PATH command, appending a PATH command to the end of AUTOEXEC.BAT wipes out the path you have previously defined. That's why installing a new program often means you can't use your DOS programs or run your applications from any directory until you edit AUTOEXEC.BAT yourself to get rid of the new PATH command and modify your original PATH command to include the new directory.

If something goes wrong

If you make an error in AUTOEXEC.BAT that keeps DOS from running properly, you can't use your system by starting it in the normal way because DOS runs AUTOEXEC.BAT each time you start or restart your system. Don't worry: Simply put your unaltered copy of the DOS system diskette in drive A—you *did* make an archive copy of the DOS disks, didn't you?—and restart the system. Now change (or delete) AUTOEXEC.BAT, remove the system diskette from drive A, and restart your system the usual way.

Creating batch commands that operate on variable data

You've probably noticed that most DOS commands require that you specify some data on which to operate. For example, the COPY command, which takes the form

```
copy  source  target
```

requires two pieces of data: *source* and *target*. Each time you issue the COPY command, you'll specify a different *source* and *target* for it to operate on.

Just as you need to supply some data for most of your DOS commands, you'll often want to supply data for your custom batch commands. Fortunately, DOS allows you to include a special symbol in your batch file called a *replaceable parameter*. When you execute the batch file, DOS will replace this symbol with the parameter you type, along with the batch command name.

The symbol you use as a replaceable parameter is a percent sign (%), followed by a number from one to nine. If you need to use only one replaceable parameter, use the symbol *%1*. If you need to use multiple replaceable parameters, use the symbol *%1* to represent the first parameter, use *%2* to represent the second parameter, and so forth.

Creating a VIEW command

We'll illustrate how to build batch commands that operate on variable data with an easy example. For instance, let's build a batch command called VIEW, which displays one screen at a time, the contents of the file you specify. The form of this command will be

```
view  filename
```

where *filename* is the name of the file whose contents you want to view.

All the batch file needs is a single MORE command (which is explained in the letter that begins on page 2) and one replaceable parameter that identifies the file you want to view. To create the file VIEW.BAT, type the following:

```
C>copy  con  view.bat
more  <  %1
^Z
            1  File(s)  copied

C>_
```

Now, whenever you type a VIEW command, DOS will execute the file VIEW.BAT, substituting the parameter you type in place of the symbol *%1*.

Let's test this batch file by using it to display the contents of the file LETTER.TXT, which we created on page 1. Simply type

```
C>view  letter.txt
```

Immediately, DOS will respond by displaying the contents of VIEW.BAT with the parameter *letter.txt* in place of the symbol *%1*:

```
C>more  <  letter.txt
```

Then, DOS will execute the command and let you view the contents of the file LETTER.TEXT:

```
Dear  Homer,

I  know  this  comes  as  a  surprise,
but  I've  left  you  for  another  man.
He's  not  as  smart  as  you,  but  he's
very  good  looking  and  rich.

Sincerely,

Marge

C>
C>_
```

Creating a WHEREIS command

Now, let's use a replaceable parameter to create a WHEREIS command. The WHEREIS command will search through the current disk for all the files whose names contain a specified string of characters. The form of this command will be

```
whereis  string
```

where *string* is the known string of characters that appears in the file's name. The result of the WHEREIS command will be the full path name of all the files on the current drive whose names contain *string*.

As we explained in the letter on page 2, you can use the command

```
attrib  \*.*  /s  |  find  "string"
```

to find the files on the current disk whose names contain *string*. As you'll recall, this command pipes the command *attrib *.* /s*, which lists all the files on the

current disk, to the command *find* "string", which displays only those files whose names contain *string*.

Now, to create the batch file WHEREIS.BAT, we'll need to use this piping technique with a replaceable parameter. To create the command, type

```
C>copy con whereis.bat
attrib \*.* /s | find "%1"
^Z
            1 File(s) Copied

C>_
```

Now, whenever you type a WHEREIS command, DOS will execute the file WHEREIS.BAT, substituting the parameter you type in place of the symbol *%1*.

Let's test this batch file by using it to find all the files whose names contain the string WK1. Simply type

```
C>whereis WK1
```

Immediately, DOS will respond by displaying the contents of WHEREIS.BAT with the parameter *WK1* in place of the symbol *%1*:

```
C>attrib \*.* /s | find "WK1"
```

Then, DOS will execute the command and display the full path name of all the appropriate files:

```
A           C:\MISC\BROWN.WK1
A           C:\123\SALES.WK1
A           C:\123\SHEET1.WK1
A           C:\123\BUDGET90.WK1
A           C:\123\PMTS.WK1

C>
C>_
```

(By the way, to eliminate the "double prompt" that DOS displays after executing the batch command, simply type [Ctrl]z at the end of the batch file's last line, instead of typing it onto a line by itself.)

Conclusion

Just as many DOS commands operate on the data you supply, many batch commands you create will need to operate on some supplied data. Thanks to replaceable parameters, which consist of a percent sign and a number from one to nine, you can easily create flexible batch files. In this article, we've introduced replaceable parameters, and we've demonstrated their use by creating a VIEW and a WHEREIS command.

An introducion to Edlin—DOS' line editor

Continued from page 1

Don't worry if you enter a line containing a mistake. As you'll learn later, it's easy to go back and edit a line or replace it with a new line. After you've typed your last line of text, press either [Ctrl][Break] or [Ctrl]c. (Alternatively, you can press [Ctrl]z and press ↵.) When you do this, Edlin will display the characters ^*C* and return to the Edlin prompt

```
6:*^C

*_
```

By the way, the asterisk that appears next to a line number isn't part of your document. Edlin uses the asterisk to indicate the current line. You can think of the current line in the same way you think of the current directory. Edlin will operate on that line unless you specify otherwise.

Displaying lines

After you've entered lines of text with Edlin, you may want to display the lines you've typed by using Edlin's List command, which is abbreviated with the letter *l*.

When you type a line number and the letter *l*, Edlin displays that line, followed by the rest of the lines in the file. For instance, when you type the number *1* followed by the letter *l*, Edlin will display all the lines you've entered so far:

```
*1l
   1: Dear Homer,
   2: I know this comes as a surprise,
   3: but I've left you for another man.
   4: He's not as smart as you, but he's
   5: very good looking and rich.
*_
```

(By the way, if more than 23 lines (one screen) follow the line whose number you type before the letter *l*, Edlin will display only the first 23 lines.)

If you want to display a specific range of lines, precede the letter *l* with two numbers, separated by a comma, that specify the first and last lines you want to see. For instance, to view lines 2 through 4, type *2,4l*, like this:

```
*2,4l
   2: I know this comes as a surprise,
   3: but I've left you for another man.
   4: He's not as smart as you, but he's
*_
```

Inserting lines

Once you've created a few lines in your file, you can insert new lines anywhere you want. To insert a new line, simply precede the letter *i* with the number of the line above which you want the new line to appear. For instance, suppose you want to insert a blank line above line 2. To do this, first type the command *2i*, like this:

```
*2i
    2:*_
```

Notice that Edlin displays the prompt *2:**, indicating that the text you enter will become the new line 2. Now, press ↵ to insert a blank line:

```
    2:*
    3:*_
```

As you can see, pressing ↵ causes Edlin to bring up the prompt for line 3. Edlin will continue inserting the lines you type until you press [Ctrl]c to return to the Edlin prompt:

```
    3:*^C
*_
```

If you want to add lines to the bottom of the file, precede the letter *i* with either a number greater than the last line number, or the character #, like this:

```
*#i
    7:_
```

Once again, you can type the lines you want to add, then press [Ctrl]c to return to the Edlin prompt:

```
    7:*
    8:*Sincerely,
    9:*
    10:*Marge
    11:*^C
*_
```

After you've made a few insertions, you'll want to use the command *1l* to display all the lines in the file:

```
*1l
    1:  Dear  Homer,
    2:
    3:  I  know  this  comes  as  a  surprise,
    4:  but  I've  left  you  for  another  man.
    5:  He's  not  as  smart  as  you,  but  he's
    6:  very  good  looking  and  rich.
    7:
    8:  Sincerely,
    9:
    10:  Marge
*_
```

Saving the file and returning to DOS

When you're finished using Edlin to work on a file, simply enter the command *e* (for *end*). When you do this, Edlin will save the changes you've made to disk and will return you to the DOS prompt:

```
*e

C>_
```

Printing the file

To print the contents of any text file, simply use the command

```
type  filename  >  prn
```

where *filename* is the name of the file whose contents you want to print. For example, to print our sample file LETTER.TXT, you use the command

```
C>type  letter.txt  >  prn
```

That's all there is to it. If you're using a laser printer and your file is shorter than a full page, you must send a form feed command to the printer to eject the printed page. To do this, either take the printer off line and press its Form Feed button, or use the technique described in the article on page 3.

Editing an existing text file

As we mentioned earlier, you can use Edlin either to create a new text file, or to edit an existing file. To edit our sample file after you've saved it and returned to DOS, simply issue the command

```
C>edlin  letter.txt
```

Since the file LETTER.TXT already exists, Edlin will load a copy of that file into memory and display the message

```
End  of  input  file
*_
```

At this point, you'll probably want to use the command *1l* to list the contents of the file. Then, you can begin modifying the contents of the file.

Deleting lines

To delete a line, simply type the number of the line you want to delete and the letter *d* (for *Delete*). For instance, to delete line 9 from our sample file, type

```
*9d
*_
```

If you want to delete several lines, precede the letter *d* with two numbers, separated by a comma, that specify the first and last lines you want to delete. For instance, to delete lines 1 through 4, type

```
*1,4d
*_
```

After deleting these lines, you can verify that they are gone by displaying all the lines in the file with the command 1l:

```
*1l
    1: He's not as smart as you, but he's
    2: very good looking and rich.
    3:
    4: Sincerely,
    5: Marge
*_
```

Aborting an editing session

If you make changes to a file, then decide you really do not want to save those changes, you can use the command *q* (for *Quit*) to abort an editing session. For example, to abort the editing session for LETTER.TXT, type

```
*q
```

Immediately, Edlin will respond

```
Abort edit (Y/N)? _
```

Type *y* to confirm that you want to abort the session without saving your changes. Edlin will then return you to the DOS prompt. If you accidentally issue the Quit command and want to continue editing the file, type *n* to return to the Edlin prompt.

After you abort the editing session, you can verify that Edlin did not save your changes by issuing the command

```
C>type letter.txt
```

When you do this, DOS will display the entire contents of that file:

```
Dear Homer,

I know this comes as a surprise,
but I've left you for another man.
He's not as smart as you, but he's
very good looking and rich.

Sincerely,

Marge
```

Conclusion

In this article, we've introduced you to Edlin—DOS' built-in text processor. The basic techniques we've shown you in this article allow you to create and edit your own text files. In future issues of *Inside DOS*, we'll show you some advanced techniques you can perform with Edlin, including editing a line, searching and replacing characters, and copying and moving lines.

Letter from the Editor

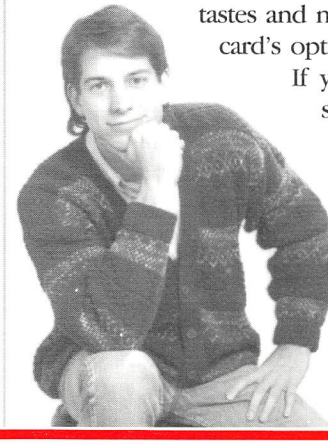
In our premier issue of *Inside DOS*, we asked you to fill out the Ratings card in the center of the journal and to mail it back to us.

Well, we've received hundreds of responses. The comments we've received vary widely from "The technical level of the journal is perfect—keep it where it is" to "The journal is way too basic—I want more advanced articles." Although we're not surprised that our readers vary widely in their level of technical expertise, we are amazed at the number of you who've requested more advanced articles.

For this reason, we are providing a card in the middle of this issue that asks you to make a simple choice:

1. I like the journal just the way it is—don't raise the technical level substantially.
2. The journal is too basic for me—I want more advanced articles.

Please give us your input! If you don't vote, you'll make it difficult for us to provide a product that best suits your tastes and needs. Just choose one of the card's options and drop it in the mail. If you want, use the provided space on the card to give us your comments.



Thank you,

Mark W. Crane

Editor-in-Chief

Using MORE with a write-protected diskette
Continued from page 2

Notice that this technique requires you to precede the file name *readme.txt* with the name of the drive on which the file is stored (*a:*), since this file is no longer stored on the default drive.

Alternatively, you can solve the write-protect problem by using a different form of the MORE command. To view the file README.TXT one screen at a time, you can use the command

```
A>more < readme.txt
```

This command allows DOS to feed the contents of the file README.TXT to the MORE command directly, without having to create a temporary file.

In the article that begins on page 8, we'll show you how to create a batch file command named VIEW that uses this form of the MORE command to view the contents of a text file. Once you create this batch file, you can use the command

```
view filename
```

to generate a screen-by-screen view of the file whose name you specify.

Finding lost files with ATTRIB and FIND

Q In the lead article of your May issue, you explained a technique for finding lost files that involves the CHKDSK and FIND commands. The command you recommended was

```
chkdsk /v | find "string"
```

where *string* is a known string of characters that appears in the name of the file(s) you want to find.

Although your technique will work without a hitch most of the time, I prefer to use the ATTRIB command instead of the CHKDSK command for this purpose. Here's why: Before the CHKDSK command generates the list of file names, it examines the status of the current disk. If CHKDSK finds any bad clusters, it won't display the list of file names. Instead, it will display a message

```
5 lost clusters found in 5 chains.
Convert lost chains to files (Y/N) ?
```

and wait for a response. In this situation, the command

```
chkdsk /v | find "string"
```

won't do the trick.

To get around this problem, I generate the list of file names with the command

```
attrib \*.* /s
```

This command tells DOS to list the full path name of every file stored on the current disk, along with the attributes of each file. Consequently, you can find lost files by piping the ATTRIB command to the FIND command, like this:

```
attrib \*.* /s | find "string"
```

This command should always work, regardless of the condition of the disk on which the files are stored.

Clifford Stevens
Dallas, Texas

A You're right, Clifford. The ATTRIB command is better suited for generating the full path names for all the files on disk. People who aren't familiar with the ATTRIB command, however, will notice that ATTRIB places the letter *A* next to the names of the files whose archive bit is turned on, and the letter *R* next to the names of the files whose read-only bit is turned on. We will discuss in detail the ATTRIB command along with a file's archive bit and read-only bit in a future issue of *Inside DOS*.